Package 'bigbang'

October 24, 2026

Type Package

Title Build 'Tidyverse'-Style Meta-Packages from Local Package Files

Version 0.6.0

Description Turns a curated set of package archives (.tar.gz, .zip) into one meta-package in the style of the 'tidyverse', so that a group of interdependent packages can be distributed and installed as a single unit. The generated meta-package records the exact archive versions it was built from and installs its components in dependency order, so that whoever receives it does not have to work out which package to install first.
The component archives are copied into the generated meta-package, so it is the only artifact that has to be distributed and no directory has to be agreed on between machines. Resolves dependencies by building a graph with topological ordering and cycle detection, classifies them as local or external, and detects implicit dependencies by scanning source code. Installation needs no repository access unless a component depends on a package that only exists in one, which suits teams working behind institutional firewalls. Generates the complete meta-package scaffold, including installation helpers, vignettes and documentation.

License GPL (>= 3)

URL https://sebollin.github.io/bigbang/, https://github.com/sebollin/bigbang

BugReports https://github.com/sebollin/bigbang/issues

Encoding UTF-8

Language en

VignetteBuilder knitr

Depends R (>= 4.1.0)

Imports brio, glue, tools, utils, whisker

Suggests devtools, covr, knitr, pkgbuild, pkgload, rmarkdown, roxygen2, testthat (>= 3.0.0), withr

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

NeedsCompilation no

Author Sebastián Lucas [aut, cre] (ORCID:
<https://orcid.org/0009-0009-9068-0276>),
Richard Detomasi [ctb] (ORCID: <https://orcid.org/0000-0002-6725-0261>,
Suggested the initial idea of a metapackage builder)

Maintainer Sebastián Lucas <sebalucas@gmail.com>

Repository CRAN

Date/Publication 2026-10-23 23:40:02 UTC

Contents

create_metapackage 2
diagnose_dependencies 11
install_local_pkg 12
print.bigbang_artifact_scan 14
print.bigbang_install_result 14
print.bigbang_result 15
scan_bigbang_artifact 15

Index **18**

`create_metapackage` *Build a local meta-package*

Description

Creates the full structure and files of a meta-package that installs, manages and loads a set of locally stored R packages, resolving the dependencies between them with a graph-based (topologically ordered) approach.

Usage

```
create_metapackage(
  name,
  packages,
  pkg_dir = NULL,
  ext = ".tar.gz",
  version = "0.1.0",
  dest_dir,
  reexport = FALSE,
  document = TRUE,
  verbose = getOption("bigbang.verbose", interactive()),
  authors =
   "person('First', 'Last', email = 'first.last@example.com', role = c('aut', 'cre'))",
  description = "Local Package Metapackage",
```

```
  license = "MIT + file LICENSE",
  additional_deps = NULL,
  ignore_deps = NULL,
 import_deps = c("data.table", "dplyr", "ggplot2", "readr", "tibble", "tidyr", "xts",
    "zoo"),
  force_deps = NULL,
  debug = FALSE,
  workflow = NULL,
  include_archives = TRUE,
  tolerate = character(),
  dry_run = FALSE,
  on_component_error = c("abort", "skip"),
  update = FALSE,
  install_upgrade = c("newer", "always", "never"),
  reexport_prefer = character(),
  reexport_exclude = character(),
  recover = FALSE,
  governance = FALSE
)
```

Arguments

name	Character. Name of the meta-package to create (must not contain underscores _).
packages	Character vector. Archive paths or stems of the local packages to include. An existing file is always used as a path; otherwise the element is resolved as a stem in pkg_dir, e.g. "myPackage_1.0.0". A bare package name such as "myPackage" resolves when exactly one archive in those directories declares that Package identity. Zero matches use the usual unresolved-archive error; multiple matches are an ambiguity error. Supported archives that cannot be read during this identity search are excluded with a warning that names the archive. Existing paths may come from different directories. A single existing text file without a recognized archive extension is treated as a manifest, with one component per line; relative paths in that file are resolved relative to the manifest directory, absolute paths and ~ paths are used as written, and bare archive filenames may also be found in pkg_dir.
pkg_dir	Character. Optional directory or directories containing local archives used to resolve stems and bare package names. It is not needed when every packages element is an existing archive path.
ext	Character. Fallback archive extension for stems. Defaults to ".tar.gz"; each existing archive path keeps its own extension.
version	Character. Version of the meta-package. Defaults to "0.1.0".
dest_dir	Character. Required destination directory. The function writes the generated meta-package exclusively inside this directory; there is no default path. Use tempdir() for disposable output.
reexport	Logical flag retained in its original position for positional-call compatibility. The default FALSE attaches installed components as usual. With TRUE, explicit

exports read from each component's NAMESPACE are exposed through read-only active bindings. Components are never added to `Imports` or `Depends`, so the generated package still installs offline without them. Evaluating a binding never throws: if a component is absent, cannot be loaded, or is an older installation that no longer exports the symbol, it returns a callable placeholder. Calling it reports the component, installed version, missing export, and the `<name>_install()` call that repairs the installation. Namespace inspection is safe for the same reason. For non-function exports, access therefore returns the placeholder instead of the object until the component is installed. The same binding then works without reloading the metapackage. Only explicit `export()` directives become bindings. S4 classes and methods remain available by loading their component package. Non-syntactic explicit export names are quoted in the generated NAMESPACE. An object restored with `readRDS()` does not load a component by itself, so base R cannot dispatch that component's S3 method until it is loaded.

`document` Logical. If `TRUE`, runs `devtools::document()` automatically. Defaults to `TRUE`. The planned `man/<name>_*.Rd` and internal-helper Rd filenames are reserved for generated documentation; custom Rd files should use different names. A successful documentation run may adopt a reserved filename into the generation manifest, after which a later update with `document = FALSE` removes it as generated output.

`verbose` Logical. If `TRUE`, shows verbose messages. The default follows `getOption("bigbang.verbose", interactive())`.

`authors` Character. Content for the `Authors@R` field of DESCRIPTION.

`description` Character. Description of the meta-package.

`license` Character. License of the meta-package.

`additional_deps`
Character vector. Extra dependencies to add on top of the ones declared by components. Source-code guesses are diagnostic by default; use this argument when a guessed dependency should bind in the generated package.

`ignore_deps` Character vector. Dependencies to ignore even if detected.

`import_deps` Character vector. Packages that should go in the `Imports` field of DESCRIPTION rather than `Depends`. Imports are not attached when the user calls `library()` on the meta-package, but remain available via `::` (e.g. `dplyr::filter()`), reducing name clashes in the user's workspace.

`force_deps` Character vector. Exact package names to use as dependencies, bypassing automatic detection. If supplied, only these are used as the meta-package's implicit dependencies.

`debug` Logical. If `TRUE`, emits detailed debugging messages. Defaults to `FALSE`.

`workflow` Optional named character vector mapping ordered stage labels to component package names. When supplied, every component must appear once and a pipeline vignette skeleton is generated.

`include_archives`
Logical. If `TRUE`, the default, the component archives are copied into `inst/archives/` of the generated meta-package, so that the meta-package is the only artifact that

has to be distributed and <meta>_install() works with no arguments, without any path being agreed on beforehand. Components still install only where they can: a Windows binary archive is refused on other platforms. Shipping the archives also means redistributing them, so their licenses have to allow it, and it makes the generated tarball as large as its components: CRAN prefers source tarballs under 10 MB and does not accept binary executables in them, which matters only if a generated meta-package is ever submitted there. Set it to FALSE when the archives stay in a shared location that recipients can reach; then <meta>_install() requires an explicit pkg_dir.

tolerate
: Character vector of explicitly named validation relaxations. Use "filename_mismatch" to silence filename-versus-DESCRIPTION mismatch warnings, or "unincluded_local_dep" to turn an available-but-unincluded local dependency error into a warning. With the latter relaxation, the generated metapackage does not ship that dependency: the recipient must provide it through pkg_dir or a repository with cran_deps = "install". Unknown names are errors. Each applied relaxation is recorded in the returned tolerated table.

dry_run
: Logical. If TRUE, resolves and validates components and returns the planned generation without creating dest_dir or writing a project. For an update, sibling journal reconciliation is also planned and reported with its paths and actions without changing those folders.

on_component_error
: Character policy for component-level failures: "abort" (default) stops generation, while "skip" omits the failed component and transitively omits components that depend on it. When a failed archive still exposes its DESCRIPTION, propagation uses its declared Package; otherwise the filename-derived name is used and the limitation is reported. If that fallback name differs from Package, a dependent may fail on the recipient. During an update, omitted inputs never authorize deletion of a previously shipped archive. When the old component cannot be identified unambiguously, archive reconciliation is deferred until a clean update rather than risking the only surviving copy.

update
: Logical. If TRUE, update a previously generated project only when its bigbang manifest is present and all generated files are unchanged. A planned file absent from both the manifest and the project is a new generated file and is added. A planned file already present outside the manifest is treated as user content and makes the update fail without touching it. Files outside that manifest are never touched. Updates are refused when the generated project root, a manifest file, or any path component inside the project is a symbolic link, so writes cannot escape the project tree. Generated files no longer in the plan are reported in removed_files. Removing a component also removes its shipped archive, which may be the last available copy. Before changing the project, an update backs up every generated file and its manifest. A failed update restores that state so the same update can be retried. Updates take an exclusive project lock across preparation, reconciliation, generation, rollback, and journal publication. A second session preserves a live preparation and reports recover = TRUE as the next action until the owner is proven to have finished. Documentation files requested by document = TRUE follow the same rule: absent planned files are added, while existing untracked files are refused. See document for the reserved generated-documentation filenames.

`install_upgrade`
: Character default upgrade policy emitted in the generated installer function: "newer", "always", or "never". This controls whether a generated installer keeps newer installed versions, reinstalls every component, or skips archive inspection.

`reexport_prefer`
: Named character vector mapping symbols to the one component that should provide them when `reexport = TRUE`, for example `c(filter = "componentb")`. Names and values must be non-empty and each named component must be included and export the mapped symbol.

`reexport_exclude`
: Character vector of symbols that must not be re-exported. Symbols are validated against the explicit exports of the included components and cannot also appear in `reexport_prefer`.

`recover`
: Logical. With `update = TRUE`, request recovery after the durable journal owner has been confirmed finished, or when a file has user content that is neither the original nor an intended update value. Unknown content is copied byte for byte to a new preserved directory beside the project before recovery; that directory is reported and is never removed automatically. A proven-live owner still blocks mutation; an uncertain owner can be reclaimed only with `recover = TRUE`. Defaults to `FALSE`.

`governance`
: Logical. If `TRUE`, generate English contribution, conduct, and package-inclusion-request templates. Defaults to `FALSE`.

Details

The function performs the following steps:

1. Creates the basic R package structure (`R`, `man`, `vignettes`, etc.).
2. Detects dependencies between packages, both explicit (from DESCRIPTION) and possible implicit uses (found by scanning executable source tokens). The latter are reported for diagnosis and are not hard dependencies unless explicitly supplied through `additional_deps` or `force_deps`.
3. Generates DESCRIPTION and NAMESPACE with the appropriate dependencies.
4. Creates a basic vignette documenting the meta-package.
5. Generates R files with functions to install and load the component packages:
   - `<name>_install()`: installs the component packages from the local archives.
   - `<name>_attach()`: attaches the components that are already installed.
   - `<name>_detach()`: detaches all the meta-package's components.
   - `<name>_packages()`: lists the included packages.

Installation is **explicit**: calling `library(<meta>)` attaches the components that are already installed and reports which ones are missing, but does not install anything or delete any files. To install the components from the local archives, the user calls `<meta>_install()`. Installation resolves dependencies with a graph-based topological ordering that also detects circular dependencies.

Generation validates every supplied component and its dependency graph eagerly before writing the metapackage. This hard validation protects an artifact that will be distributed to another machine. The installer is more tolerant: when an already installed component does not need to be changed, it can retain that installation without reading an archive that will not be used.

Value

Invisibly, a `bigbang_result` containing the generated path, component archives, dependency classification, applied tolerations, files removed by the call, documentation status, the `reexports` table, `reexport_excluded` symbols, and whether an interrupted update was recovered. Recovery details include any directory used to preserve unknown user content and the sibling-journal reconciliation plan.

Validation strictness

During generation, validations that protect the recipient cannot be disabled: malformed or unsafe archives, invalid component metadata, duplicate components, cycles, and unsatisfied local version constraints remain hard errors. Checks about project tidiness can be relaxed individually through `tolerate`; there is no switch that disables validation as a whole. bigbang does not run `R CMD check` on component packages, so component warnings and notes do not prevent generation.

Portable archive names

Archive entry names are validated strictly by default. The option `bigbang.archive.portable_names` defaults to `TRUE` and rejects Windows device names, Windows-invalid characters, trailing dots or spaces, and case-only collisions on every system. Set it to `FALSE` only when every recipient uses Linux or macOS. On non-Windows systems this relaxes only the Windows-specific rules; case-only collisions are relaxed only when a real probe of the extraction filesystem shows that it distinguishes case; an inspection without an extraction destination defers that decision until extraction. Components made only of dots and spaces are always rejected. Links, special entries, absolute or parent-traversal paths, empty components, backslashes, control characters, invalid UTF-8, and size limits are never relaxed. Windows always applies the strict Windows rules, regardless of the option. Relaxed entries produce warnings and the resulting package cannot be installed on Windows, or on a filesystem that does not distinguish case. See `scan_bigbang_artifact()` for the same archive-name policy and resource limits used by the reader. Component source directories are built in a temporary directory with the optional pkgbuild package; passing an already built archive avoids that optional dependency.

Component installation

The generated meta-package installs component packages only when the user explicitly calls `<meta>_install()`. Loading it with `library()` never installs packages. By default, the generated installer does not access a repository.

With `include_archives = TRUE`, the default, the component archives travel inside the generated meta-package and `pkg_dir` defaults to `system.file("archives", package = "<meta>")`. That default is resolved when the installer is called, so it points at the library of whoever installed the meta-package: recipients need nothing beyond the meta-package itself, and no path has to be agreed on between machines. Network access is needed only when a component depends on a package that must come from a repository, which happens exclusively under `cran_deps = "install"`.

Loading the generated meta-package attaches installed components, so their exported functions can be called directly or through `component::function()`. With `reexport = TRUE`, explicit component exports are instead exposed through read-only active bindings in the meta-package namespace. This does not add components to `Imports` or `Depends`: loading remains possible without them, and

a binding resolves the component on every access. Evaluating a binding never throws: if a component is absent, cannot be loaded, or is an older installation that no longer exports the symbol, it returns a callable placeholder. Calling it reports the component, installed version, missing export, and the `<name>_install()` call that repairs the installation. This also keeps namespace inspection safe. Only explicit `export()` directives are rebound; S4 classes and methods are used through the loaded component namespace. An object restored with `readRDS()` cannot load a component by itself, so base R cannot dispatch that component's S3 method until the component has been loaded.

Re-export collisions

When more than one component exports a symbol, `reexport_prefer` chooses its provider explicitly and `reexport_exclude` removes it from the generated namespace. Every collision requires one of those options because static source analysis cannot prove that two exported objects are the same at runtime. The analysis remains as a diagnostic with `probable_same_object`, `distinct_definitions`, or `undetermined`, including ordered file, line, import, and parse reasons. For a preferred `probable_same_object`, `<name>_install()` verifies the installed owners in a clean R subprocess whose destination library is first in `.libPaths()`. If that subprocess cannot run, the result is explicitly unverified and never reports a false identity. A namespace already loaded from another library is reported before the clean verification starts. Missing owners remain unverified and the verification is retained in the returned result. The diagnostic is a help, not the guarantee: the guarantee is the explicit `reexport_prefer` or `reexport_exclude` decision plus that verification. Calling `library(<meta>)` alone does not verify installed owners. The scanner is deliberately conservative and can count a never-forced `delayedAssign`, an `if (FALSE)` branch, or a `reg.finalizer()` body; this overcount does not weaken the explicit decision and installation-verification guarantee. `<name>_conflicts()` repeats that check on request. Its masking-conflict names remain ordinary symbols; use `<name>_reexport_verification(conflicts)` to access the verification attribute without a name collision. If `on_component_error = "skip"` omits a component required by a preferred binding or an import source, generation errors with an actionable skipped condition instead of creating a binding to a component that will not travel with the metapackage. With `reexport = TRUE`, `<name>_conflicts()` retains the masking-conflict list from earlier releases and stores its installed-owner table as an attribute. The accessor keeps the same `<name>_reexport_verification` class when it has zero rows.

Interrupted updates

Before an in-place update mutates the project, bigbang assembles a durable journal beside it in a private `.<name>.bigbang-update.armando-*` folder. The marker is written before the backup, and the complete folder is renamed to `.<name>.bigbang-update` only after every hash has been verified. Every later file write or removal records its intention first. Generated files, shipped component archives, catalogs, `.Rbuildignore`, and the final manifest are replaced atomically. On Windows the guarantee is that a file is old, new, or temporarily absent with a journal backup. Roxygen runs in a staging copy and only its known outputs are promoted atomically to the project.

Updates also publish `.<name>.bigbang-update.lock` atomically from a sibling temporary folder that already contains a complete `owner.rds`. A published lock therefore always has an owner. Reclaiming an orphan first atomically renames it to a unique discarded name; only the process that wins that rename may publish a replacement, and it rechecks the owner before doing so. Lock disposition is owner-first. For the published lock, a proven live owner blocks every caller; an uncertain owner blocks without `recover = TRUE` and is reclaimable only with `recover = TRUE`; a proven dead owner is reclaimable. For a discarded lock, a proven live `owner.rds` is restored when

the lock name is free or blocks on its PID when it is occupied. It is never deleted. An uncertain discarded owner follows the uncertain-lock rule. Only after the discarded owner is proven dead does the claimant decide the outcome: a live claimant blocks, an uncertain claimant needs `recover = TRUE`, and a dead claimant may be discarded. `owner.rds` and `claim.rds` are removed only when their bytes still have the digest observed for that decision; mismatches are preserved by setting the entry aside. No claim is written before the owner has been re-read immediately before publication. The update also revalidates its published owner before creating the journal, recording each intent, and completing an irreversible step. If the owner changed, it aborts before the next mutation. A discarded entry can therefore contain a claimant record, but that record is never allowed to override a live owner. For a `.lock.armando-*` entry, a live owner stays in place and blocks; an uncertain owner stays in place without `recover = TRUE` and is set aside with `recover = TRUE`; a dead or missing owner is set aside. A symbolic link at the published lock name is reported as a link without an update-running claim; with `recover = TRUE` the link itself is renamed aside and its target is not followed. A regular file or other user entry at the lock name is atomically set aside as `.<name>.bigbang-apartado-*`. Lock preparations left by an interruption are recognized on the next call and set aside without deleting their bytes. These names are reserved bigbang siblings: `.<name>.bigbang-update`, `.<name>.bigbang-update.armando-*`, `.<name>.bigbang-update.lock`, `.<name>.bigbang-update.lock.armando-*`, `.<name>.bigbang-update.lock.descar` `.<name>.bigbang-update.descartado-*`, and `.<name>.bigbang-apartado-*`.

If a process dies while preparing the journal, an empty unmarked `armando-*` folder is removed; any non-empty unmarked folder is atomically set aside as `.<name>.bigbang-apartado-*` without copying or deleting bytes. The initial marker records the owner PID, host, process start token, and start time before the first backup copy. On Linux, liveness reads `/proc/<pid>` and treats a missing process as dead, `Z` or `X` in `/proc/<pid>/stat` as dead, and any other readable state as existing; the process-start token still decides identity. Without `/proc`, `kill(pid, 0)` proves existence only when it succeeds; if that probe is unavailable, `LC_ALL=C ps -p <pid>` establishes whether the PID is present or absent, and `ps -o lstart= -p <pid>` supplies the portable start token. The token source is stored (`proc` or `ps`) and mismatched sources never compare equal. A failure is dead only when `ps -p` also proves that the PID is absent; permission errors, an unavailable `ps`, and an unreadable token are uncertain. The exact policy is: dead means the process does not exist or is `Z`/`X`; alive means it exists, is not terminal, and its start token matches; live-token-conflict means it exists but the token differs; uncertain means existence or identity cannot be proved. `recover = TRUE` may claim or set aside uncertain entries, but never overrides a proven live owner. A process of another user is therefore never inferred dead from `EPERM`. Journal disposal first writes an atomic tombstone with the exact relative-path and MD5 inventory of the entries bigbang wrote, then renames the folder to `.<name>.bigbang-update.descartado-*`; cleanup can therefore resume after another interruption. Cleanup checks every file recursively and removes it only when its relative path and MD5 match the inventory; it removes an inventory directory only after it is empty. Before destructive cleanup the journal is renamed to an unpredictable private sibling after verifying it is not a link, and each deletion revalidates its ancestors and MD5 immediately before `unlink()`. Any file, directory, or symbolic link that cannot be proved to be in the inventory causes the whole discarded folder to be set aside atomically and reported, so the update continues without deleting user bytes. The tombstone has a digest recorded beside it before the rename; a missing or changed digest is set aside rather than trusted. A discarded folder without a valid tombstone is set aside when non-empty; an empty one is removed as an interrupted cleanup shell. A matching name and manifest are required before a discarded folder is cleaned. A stale generation or another project is therefore set aside beside the current project and never blocks a later update. A file with the same path and MD5 as the inventory is an unavoidable limit: its bytes are identical, so deleting

it loses no content, but the journal cannot prove who created it. The tombstone and its digest are local journal state, not a cryptographic signature; treat the journal as bigbang's private territory. A process of the same user with write permission can forge `owner.rds`, `marker.rds`, or `state.rds`; that is outside this integrity model. R has no `unlinkat()`/`O_NOFOLLOW`, so a same-user process that actively replaces journal directories during discard remains an integrity boundary; the remaining race is the interval between the last revalidation and `unlink()`. As a cheap consistency check, an armed journal is recoverable only when the owner fields in `state.rds` match those in `marker.rds`; otherwise the journal is set aside and is never used for rollback.

The journal is designed to survive process interruptions such as SIGKILL, an R error, or Ctrl-C. It does not promise fsync durability against an OS or power shutdown. The next `create_metapackage(update = TRUE)` call examines it before validating the generation manifest. The marker identifies the metapackage and old-manifest hash rather than an absolute path, so moving the project together with its journal remains recoverable. Renaming a project is not supported: generated file names contain the metapackage name. Rename the project and its journal back to `<name>` before updating. A byte-for-byte copy placed at the same path and name as the moved original is indistinguishable from that original; the journal consequently treats it as the project. If the original project still exists beside a copied journal, the journal is not adopted or changed. A partial tombstone temporary is set aside after the owner is confirmed dead, and recovery continues. An already completed update is recognized by its new manifest; otherwise a dead owner's changes are rolled back and the requested update continues. On POSIX systems liveness uses the PID and, where Linux `/proc` exposes it, the process start time. Windows is never probed with the process-termination helper because that operation terminates a process. A dry run evaluates and reports the lock as free, live, orphaned, or uncertain without acquiring, reclaiming, renaming, or deleting any lock entry.

Automatic recovery proceeds only when every affected path contains its original bytes, intended bytes, or an expected absence. Other content raises `bigbang_error_interrupted_update`; `recover = TRUE` preserves it outside the project before rollback. Recovery is idempotent, so another interruption can be recovered by a later call. `dry_run = TRUE` reports the pending action and leaves the project and every sibling journal folder untouched. A handled error uses this same journal for immediate rollback and retains it if verification cannot finish. Documentation generation failures in the staging copy are warnings; a failure while promoting any documentation output aborts the update and rolls the complete project back through the journal.

Requirements

- Each component must be an existing archive path or a stem resolvable in one of the optional `pkg_dir` directories; `ext` is only a fallback for stems.
- Files in the supplied archive directories that cannot be read are excluded from the inventory with a warning. A requested component still fails validation, while an unreadable file matching a declared dependency is reported as an unavailable local archive.
- Automatic documentation (`document = TRUE`) requires the `devtools` package.

Examples

```
archives <- system.file("extdata", package = "bigbang")
destination <- tempfile("bigbang-example-")
dir.create(destination)

result <- create_metapackage(
```

```
  name = "toyverse",
  packages = "toycomponent_0.1.0",
  pkg_dir = archives,
  dest_dir = destination,
  document = FALSE,
  verbose = FALSE,
  import_deps = character(),
  force_deps = character()
)
list.files(result$path)

unlink(destination, recursive = TRUE)
```

`diagnose_dependencies` *Diagnose implicit dependencies of local packages*

Description

Scans local packages for references to the recommended packages 'Matrix' and 'class', which can cause `R CMD check` failures when they are used implicitly but not declared as dependencies.

Usage

```
diagnose_dependencies(packages, pkg_dir = NULL, ext = ".tar.gz")
```

Arguments

`packages`	Character vector. Archive paths or stems to examine, e.g. `"conexiones_0.8.3"`.
`pkg_dir`	Character. Directory or directories containing local archives (`.tar.gz`, `.zip`, etc.).
`ext`	Character. Archive extension. Defaults to `".tar.gz"`.

Details

Extracts and scans the R source of each package for patterns that suggest implicit use of 'Matrix' or 'class'. Useful for debugging `R CMD check` errors such as "there is no package called 'Matrix'" even when the package does not appear to use it directly.

Value

A named list with one entry per local package, each a list with two elements:

matrix_refs Character vector of references to 'Matrix', with file and line.

class_refs Character vector of references to 'class', with file and line.

Examples

```
archives <- system.file("extdata", package = "bigbang")
res <- diagnose_dependencies(
  packages = "toycomponent_0.1.0",
  pkg_dir = archives
)
res[["toycomponent_0.1.0"]]
lapply(res, function(x) x$matrix_refs)
```

`install_local_pkg` *Install a local package together with its dependencies*

Description

Installs a package from a local archive. Dependencies available as local archives are installed recursively; missing non-local dependencies follow the explicit `cran_deps` policy. ZIP archives containing `Meta/package.rds` are treated as Windows binaries, while other ZIP archives are unpacked and installed as source packages.

Usage

```
install_local_pkg(
  package,
  pkg_dir = NULL,
  ext = ".tar.gz",
  repos = NULL,
  cran_deps = c("skip", "error", "install"),
  verbose = getOption("bigbang.verbose", interactive()),
  force = FALSE,
  upgrade = c("newer", "always", "never"),
  lib = .libPaths()[[1L]]
)
```

Arguments

`package`	Character. An existing archive path, or a package stem such as `"uspr_0.8.5"` to resolve in `pkg_dir`. An existing file is always treated as a path; only a non-existing element is resolved as a stem. A bare package name such as `"uspr"` resolves when exactly one archive in those directories declares that `Package` identity; multiple matches are an ambiguity error. Supported archives that cannot be read during this identity search are excluded with a warning that names the archive.
`pkg_dir`	Character. Optional directory or directories containing local archives. It is not needed when `package` is an existing path.
`ext`	Character. Fallback archive extension for stems; existing paths keep their own extension.

repos	Character. Repositories used only when cran_deps = "install".
cran_deps	Character. Policy for missing non-local dependencies: "skip" (the default) never accesses the network, "error" fails without accessing it, and "install" attempts installation from repos.
verbose	Logical. Whether to emit progress and summary messages. The default follows getOption("bigbang.verbose", interactive()).
force	Logical. Reinstall every local archive. This is a convenience alias for upgrade = "always".
upgrade	Character. Installed-version policy: "newer" installs only when the archive is newer, "always" reinstalls, and "never" keeps any installed version. Explicitly combining force = TRUE with a value other than "always" is an error.
lib	Character. Library in which the local component must be installed and verified when supplied explicitly. When omitted, the legacy lookup considers all of .libPaths() and installs into its first entry. Non-local dependencies may be available either in lib or anywhere on .libPaths().

Details

Archive entry names follow the bigbang.archive.portable_names option described in scan_bigbang_artifact(). It is TRUE by default; set it to FALSE only when every recipient uses Linux or macOS. Relaxed names warn during installation and remain unavailable on Windows or on filesystems that do not distinguish case. Case-only collisions are decided when the archive is extracted; an earlier inspection without a destination defers that decision. Components made only of dots and spaces, Windows-specific rules, and security-critical path rules are never relaxed.

Value

Invisibly, a list describing installed, unchanged, failed, and skipped packages. Components that an upgrade policy left in place are reported in unchanged, not in installed.

Installation

This function installs packages into lib, which defaults to the user's active R library. When lib is supplied explicitly, a component found only in another library is still installed into lib; non-local dependencies are resolved from lib plus .libPaths(). When lib is omitted, an installed component found anywhere on .libPaths() retains the pre-0.3.0 behavior. Installation occurs only when the user calls the function; loading bigbang never installs packages. With the default cran_deps = "skip", it does not access the network. An installed package can be kept without reading its archive when upgrade = "never". Under the default policy, bigbang reads only the archive DESCRIPTION first; if that metadata cannot be verified for an already installed package, the installed package is kept and the reason is reported. With upgrade = "never", the shortcut takes the component identity from the archive filename because the archive is not read. Use upgrade = "newer" when the declared Package field must be checked against the installed package.

See Also

create_metapackage() for generating a meta-package with an explicit component installer; scan_bigbang_artifact() for the archive-name policy.

Examples

```
archive <- system.file(
  "extdata", "toycomponent_0.1.0.tar.gz", package = "bigbang"
)
example_library <- tempfile("bigbang-example-library-")
dir.create(example_library)
on.exit(unlink(example_library, recursive = TRUE), add = TRUE)
install_local_pkg(archive, lib = example_library, verbose = FALSE)
```

---

print.bigbang_artifact_scan

Print an artifact scan

Description

Print an artifact scan

Usage

```
## S3 method for class 'bigbang_artifact_scan'
print(x, ...)
```

Arguments

x	A bigbang_artifact_scan object.
...	Unused.

Value

x, invisibly.

---

print.bigbang_install_result

Print a local package installation result

Description

Print a local package installation result

Usage

```
## S3 method for class 'bigbang_install_result'
print(x, ...)
```

Arguments

x	A bigbang_install_result object.
...	Unused.

Value

x, invisibly.

---

print.bigbang_result *Print a metapackage generation result*

Description

Print a metapackage generation result

Usage

```
## S3 method for class 'bigbang_result'
print(x, ...)
```

Arguments

x	A bigbang_result object returned by create_metapackage().
...	Unused.

Value

x, invisibly.

---

scan_bigbang_artifact *Scan a generated metapackage for historical deletion signatures*

Description

Inspects a generated metapackage source directory, source archive, or installed package without loading it. The scanner looks for the historical V1, V2, V3, and V7 deletion signatures from the pre-release security investigation. Provenance is read from both the current generator fields and the legacy pre-rename fields so development artifacts remain classifiable.

Usage

```
scan_bigbang_artifact(path, dry_run = TRUE)
```

Arguments

path	Character scalar. Source directory, .tar.gz/.tar/.zip source archive, or installed package directory.
dry_run	Logical. Must be TRUE, the default. Automatic mutation or remediation is deliberately not implemented.

Details

Installed packages are inspected through R's internal lazy-load database API. That code is isolated in .scan_installed_lazydb() and has been exercised with R 4.6.1. Because this is an internal R format, callers should re-run the scanner tests when adopting a new R minor release.

Value

A list with the artifact type, vulnerability flag, detected signatures, evidence locations, provenance fields, and R version used for the scan.

Archive resource limits

Archive walks use `options()` values named `bigbang.archive.max_entries` (default 100,000), `bigbang.archive.max_total_bytes` (default 2 GiB), `bigbang.archive.max_metadata_bytes` (default 64 MiB), and `bigbang.archive.max_path_bytes` (default 4,096 bytes). Limits apply during both inspection and extraction; the byte limit includes archive metadata and data discarded during inspection. Metadata includes ZIP local and central records, data descriptors, the EOCD comment, every TAR header, and PAX/GNU metadata payload. Implicit directories and PAX/GNU metadata headers count toward `bigbang.archive.max_entries`. The default metadata cap leaves room for the documented maximum of 65,534 entries in a classic ZIP (ZIP64 is not supported). Entry names are compared without locale dependence, but they are not Unicode-normalized: NFC and NFD spellings remain distinct; NFD names are accepted because macOS can produce them.

The `bigbang.archive.portable_names` option defaults to `TRUE` and applies the strict Windows-compatible name policy. Set it to `FALSE` only when all recipients use Linux or macOS. Then Windows device names, Windows-invalid characters, and trailing dots or spaces may be accepted on non-Windows systems, while case-only collisions are accepted only after a real probe confirms that the extraction filesystem distinguishes case; an inspection without an extraction destination defers that decision until extraction. Components made only of dots and spaces are always rejected. Links, special entries, absolute or parent-traversal paths, empty components, backslashes, control characters, invalid UTF-8, and size limits are never relaxed. Windows remains strict regardless of the option. The result reports accepted relaxed names in `portable_name_warnings`.

Examples

```
archives <- system.file("extdata", package = "bigbang")
destination <- tempfile("bigbang-scan-example-")
dir.create(destination)

result <- create_metapackage(
  name = "toyverse",
  packages = "toycomponent_0.1.0",
  pkg_dir = archives,
```

```
  dest_dir = destination,
  document = FALSE,
  verbose = FALSE,
  import_deps = character(),
  force_deps = character()
)
scan_bigbang_artifact(result$path)

unlink(destination, recursive = TRUE)
```

Index

create_metapackage, 2
create_metapackage(), *13*, *15*

diagnose_dependencies, 11

install_local_pkg, 12

print.bigbang_artifact_scan, 14
print.bigbang_install_result, 14
print.bigbang_result, 15

scan_bigbang_artifact, 15
scan_bigbang_artifact(), *7*, *13*